CSE 141: Introduction to Computer Architecture

Who Am I?

- Beth Simon
  - PhD  Univ. of California, San Diego
  - BS   Univ. of Dayton, Ohio
- Research Interests
  - Compilers and Architectures
    - Intel IA-64 (an EPIC machine)
    - Hardware Branch Prediction Optimization Design that utilizes Predicate Define information as produced by an optimizing predicate-aware compiler
  - Performance Programming
    - Multithreading (SMT, Cray MTA)
    - Memory hierarchy optimization (caches and prog.)

Computer Architecture

- Who Am I?
  - Dr. Beth Simon
- What is the purpose of this class?
  - Span the gap between code and the physical computer
- Who are You?
  - Meet your classmates! Network!
- Class Plan
  - Syllabus review
  - How to have fun in this class
  - How to ace this class
  - Ethics

What Have I done?

- Intel production compiler group
  - Value profiling optimizations
    - Make better code based on the fact that a certain value is usually a constant (ie divide by 1)
    - Speed up SPEC2000 parser by 12%
- Published and presented several articles in top architecture and compiler conferences
- Served on several graduate admissions and undergraduate fellowship selection committees
**New and Exciting Research**

- Research Scientist at San Diego Supercomputing Center
  - Performance Modeling and Characterization Lab
  - [http://www.sdsc.edu/PMaC](http://www.sdsc.edu/PMaC)
- Using current machines, make predictions about how fast certain codes will execute on machines that the DoD/SDSC might purchase

---

**Studying Computer Science**

- Unlike other disciplines
  - not quite math
  - less repetition
  - more like engineering
    - problem analysis
    - plan of attack
    - implementation of solution
- Compare to medicine:
  - Theory
  - Practice
- This class focuses on theory
  - Some "practice" of design in lab
  - BUT it explains how your PRACTICE (programming) is implemented on an ACTUAL computer

---

**What is Computer Architecture?**

- Hardware Designer
  - thinks about circuits, components, timing, functionality, ease of debugging
  - "construction engineer"
- Computer Architect
  - thinks about high-level components, how they fit together, how they work together to deliver performance
  - "building architect"

---

**Which is faster?**

```c
for (i=0; i<N; i=i+1)  for (jj=0; jj<jj; jj=jj+B)
for (j=0; j<j; j=j+1)  for (kk=0; kk<k; kk=kk+B)
{                      for (i=0; i<N; i=i+1)
  r = 0;              r = 0;
  for (k=0; k<k; k=k+1)  for (j=jj)
    r = r + y[i][k] *  z[k][j];
    x[i][j] = r;
  }
  [1] = [1] + r;
  [i][j] = [i][j] + [i][j];
```

---
Which is faster?

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>load R1, addr1</td>
<td>load R1, addr1</td>
</tr>
<tr>
<td>store R1, addr2</td>
<td>add R0, R2 → R3</td>
</tr>
<tr>
<td>add R0, R2 → R3</td>
<td>add R0, R6 → R7</td>
</tr>
<tr>
<td>subtract R4, R3 → R5</td>
<td>store R1, addr2</td>
</tr>
<tr>
<td>add R0, R6 → R7</td>
<td>subtract R4, R3 → R5</td>
</tr>
<tr>
<td>store R7, addr3</td>
<td>store R7, addr3</td>
</tr>
</tbody>
</table>

Code PERFORMANCE is dictated by COMPUTER ARCHITECTURE

- The goal of this class is NOT to make you into electrical engineers.
- The goal of this class is to expose you to the world of computer design
  - Using a historical perspective will lead us from simpler to more complex designs
  - Will also help you understand what in a design leads to better performance.
- Though the book focuses on the architecture you MUST ALWAYS THINK:
  - HOW DOES THIS AFFECT THE PERFORMANCE OF CODE?

Who are You?

- Stand up. Introduce yourself to at least 4 people around you.
  - Name
  - How long until you graduate
  - What you want to do after graduation (besides not study)
  - Where, in the world, you would live if you could choose.

Textbook

Patterson & Hennessy, second edition of "Computer Organization, the Hardware/Software Interface"

- Exceptionally good book. We'll read most of it.
  - 3rd Edition, came out this year, look for errata
- Patterson is professor at Berkeley.
  - lead RISC project (foundation of SPARC processor)
  - lead RAID (redundant array of inexpensive disks) project
- Hennessy is professor at Stanford
  - now President of Stanford
  - co-founded of MIPS Computer Systems
- Note: same authors wrote the graduate textbook, "Computer Architecture, A Quantitative Approach"
Ubiquitous Presenter

- You will be able to access slides in this class on line.
  - You can connect to the web site during class and see all the ink I add. PASSWORD:
  - This is great for review, but STUDIES SHOW you learn better the more involved you are in your learning.
    - I suggest you print out the slides and fill them in in class.
- There will be in-class activities where you will be asked to participate using a wireless device with a web browser.

How to ACE this class!

- READ the book! (Before I talk about it in class)
  - Don't slack – catching up will be HARD
- Come to class with 2 questions written down & make sure I answer them
- At least READ the homework EARLY
- 4 credits * 3 hours/credit = __ h per w
- Believe the studies...
  - You learn 10% of what you hear.
  - You learn 20% of what you see.
  - You learn 40% of what you read.
  - You learn 80% of what you do.
  - You learn 90% of what you teach.
Chapter 1

The Overview
(read it tonight)

What is Computer Architecture?

Computer Architecture =
Machine Organization +
Instruction Set Architecture

How to Speak Computer

High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

ISA

Machine Interpretation

Control Signal Spec

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
lw $15, 0($2)
lw $16, 4($2)
sw $15, 0($2)
sw $16, 4($2)

A Review: Java on the Computer

Java Language Program

Compiler

Machine Language Program

ISA

Machine Interpretation

Control Signal Spec

ALUOP[0:3] <= IsSetReg[9:11] & MASK
void spin() {
    int i;
    for (i = 0; i < 100; i++) {
        // Loop body is empty
    }
}

The Instruction Set Architecture (ISA)

- that part of the architecture that is visible to the programmer
  - opcodes (available instructions)
  - number and types of registers
  - instruction formats
  - storage access, addressing modes
  - exceptional conditions
- It is NOT QUIETLY assembly language: but usually DARN close.

The Instruction Set Architecture

"is the agreed-upon interface between all the software that runs on the machine and the hardware that executes it.

Java vs. Byte code vs. Assembly (almost machine lang)

<table>
<thead>
<tr>
<th>Java Code</th>
<th>Byte Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void spin() { int i; for (i = 0; i &lt; 100; i++) { ; // Loop body is empty } }</td>
<td>0 iconst_0 // Push Int constant 0 1 isalw_0 // Store into local 0 (i=0) 2 galo 8 // First time through don't increment 5 inc 0 1 // Increment local 0 by 1 (i++) 8 load_0 // Push local 0 (i) 9 bpush 100 // Push Int constant (100) 11 if ecmp 5 // Compare, loop if i &lt; 100 14 return // Return so it when done</td>
</tr>
</tbody>
</table>

Assembly on an x86
Examples of ISAs
- Alpha AXP
- Intel 80x86/pentium (called x86)
- VAX
- MIPS
- SPARC
- IBM 360
- Intel IA-64 (Itanium or EPIC)
- PowerPC

Computer Organization
- Once you have decided on an ISA, you must decide how to design the hardware to execute those programs written in the ISA as fast as possible.
- There are MANY different hardware designs possible for a given ISA.
- Choices of hardware design can change:
  - For reduced cost (cheaper to produce, design)
    - $200 Walmart computer
    - Sandia National Labs ASCI Red: A Tera-Op (10^12) machine made of 4,562 Pentium Pros
  - For increased performance (hire smarter people to design)
    - Intel/HP EPIC machines
  - For reduced power requirements (hire smarter people)
    - Transmeta's Crusoe processor

The Challenge of Computer Architecture
- The industry changes faster than any other.
- The ground rules change every year.
  - new problems
  - new opportunities
  - different tradeoffs
- It’s all about making programs run faster than the next guy’s machine can run them.

Performance Trends
Starting in the 1940’s, had the transportation industry kept pace with computer performance increases, today you could travel to the east coast in 5 seconds at a cost of 50 cents.
The five classic components of computers: All “computers” can fit into this general classification

Disks & Tape

Also considered I/O devices
- Hard disks (magnetic surface on metal)
  - Very slow access time (~ 5 ms)
  - Getting inexpensive very fast
- Floppy disks (magnetic surface on mylar)
  - Cheap and convenient
- CD’s (compact disks) – optical
  - Even cheaper
  - Slow (or impossible) to write
- Magnetic tape – a dying technology

Memory: How we use it
- 2 Volatile Types:
  - Main Memory
    - Where program (and some data) is stored
    - When you turn off the power it's gone
    - Made of DRAM (usually)
  - Cache
    - A small, fast memory
    - Is a "buffer" for MM
      - Keeps a portion of MM available FASTER
      - Made of SRAM or SDRAM

Why care about power consumption?
- California's energy crisis??
  - Not really
- Heat is hard to get rid of!
  - Workstation processor might use 70 Watts
  - Limits how densely components can be packaged
- Battery power is limited!
  - Embedded processors in portable devices
Real Stuff: Chip Manufacturing

- Silicon ingot
- Blank wafers
- 30 to 50 processing steps
- Patterned wafers
- Die
- Tested dies
- Packaged dies
- Bond to package
- Cost per die = cost per wafer/dies per wafer * yield
dies per wafer = wafer area/die area
yield = 1/(1+(defects per area * die area)/2)^2

Important Stuff in Book Not Covered in Lecture

- Disks
- Networks
- CRTs and displays
- Mice
- Embedded processors
- Picture of a Pentium 4 (page 21)

Key Points

- High-performance software requires a deep understanding of the underlying machine organization.
- The instruction set architecture defines how software is allowed to use the processor. Multiple computers with vastly different organizations and performance can share an ISA.
- Most every component in a computer system falls into one of five categories.

Chapter 2: Instructions
How we talk to the computer

Pages not responsible for:
2.7 pg79-89
2.10-2.15 pg 106-134
DO READ 2.16 “Real Stuff”
**The Instruction Set Architecture**

The agreed-upon interface between:
the software that runs on a computer and
the hardware that executes it.

- **Application**
- **Compiler**
- **Operating System**
- **Instr. Set Proc.**
- **I/O system**
- **Digital Design**
- **Circuit Design**

---

**Overall goals of ISA**

- Can be implemented by simple hardware
- Can be implemented by fast hardware
- Instructions do useful things
- Easy to write (or generate) machine code

---

**The Instruction Execution Cycle**

1. Obtain instruction from program storage
2. Determine required actions and instruction size
3. Locate and obtain operand data
4. Compute result value or status
5. Deposit results in storage for later use
6. Determine successor instruction

---

**The Instruction Set Architecture**

- that part of the architecture that is visible to the programmer
  - instruction formats
  - opcodes (available instructions)
  - number and types of registers
  - storage access, addressing modes
  - exceptional conditions
Key ISA decisions

instruction length
  • are all instructions the same length?
how many registers?
where do operands reside?
  • e.g., can you add contents of memory to a register?
instruction format
  • which bits designate what?
operands
  • how many? how big?
  • how are memory addresses computed?
operations
  • what operations are provided?

Running examples
We'll look at four example ISA's:
- Digital's VAX (1977) - elegant
- Intel's x86 (1978) - ugly, but successful (IBM PC)
- MIPS - focus of text, used in assorted machines
- PowerPC - used in Mac's, IBM supercomputers, ...
- VAX and x86 are CISC ("Complex Instruction Set Computers")
- MIPS and PowerPC are RISC ("Reduced Instruction Set Computers")
  - almost all machines of 80's and 90's are RISC
    - including VAX's successor, the DEC Alpha

Instruction Length

Variable:

x86 - Instructions vary from 1 to 17 Bytes long
VAX - from 1 to 54 Bytes

Fixed:

MIPS, PowerPC, and most other RISCs:
all instruction are 4 Bytes long

Instruction Length

• Variable-length instructions (x86, VAX):
  - require multi-step fetch and decode.
  + allow for a more flexible and compact instruction set.
• Fixed-length instructions (RISCs)
  + allow easy fetch and decode.
  + simplify pipelining and parallelism.
  - instruction bits are scarce.
How many registers?

All computers have a small set of registers
   Memory to hold values that will be used soon
   Typical instruction will use 2 or 3 register values

Advantages of a small number of registers:
   - Less hardware
   - Faster access (shorter wires, fewer gates)
   - Faster context switch (when all registers need saving)

Advantages of a larger number:
   - Easier to do several operations at once

VAX - 16 registers
   R15 is program counter (PC)
   Easier! Loading R15 is a jump instruction

x86 - 8 general purpose regs
   Fine print - some restrictions apply
   Plus floating point and special purpose registers

Most RISCs have 32 int and 32 floating point regs
   Plus some special purpose ones
   - PowerPC has 8 four-bit "condition registers", a "count register" (to hold loop index), and others.
   - Itanium has 128 fixed, 128 float, and 64 "predicate" registers

Where do operands reside?

Stack machine:
   "Push" loads memory into 1st register ("top of stack"), moves other reg down
   "Pop" does the reverse.
   "Add" combines contents of first two registers, moves rest up.

Accumulator machine:
   Only 1 register (called the "accumulator")
   Instruction include "store" and "acc ← acc + mem"

Register-Memory machine:
   Arithmetic instructions can use data in registers and/or memory

Load-Store Machine (aka Register-Register Machine):
   Arithmetic instructions can only use data in registers.

Load-store architectures

Can do:
   add r1=r2+r3
   load r3, M(address)
   store r1, M(address)

⇒ forces heavy dependence on registers, which is exactly what you want in today's CPUs

Can't do:
   add r1=r2+M(address)

⇒ more instructions
   fast implementation (e.g., easy pipelining)
**Where do operands reside?**

VAX: register-memory
   Very general. 0, 1, 2, or 3 operands can be in registers
x86: register-memory ...
   But floating-point registers are a stack.
   Not as general as VAX instructions
RISC machines:
   Always load-store machines

**Comparing the Number of Instructions**

Code sequence for $C = A + B$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Accumulator</th>
<th>Register-Memory</th>
<th>Load-Store</th>
</tr>
</thead>
</table>

**Can You Do?**

\[ A = X^*Y + X^*Z \]

---

**Key ISA decisions**

- instruction length
  - are all instructions the same length?
- how many registers?
- where do operands reside?
  - e.g., can you add contents of memory to a register?
  - instruction format
    - which bits designate what??
  - operands
    - how many? how big?
    - how are memory addresses computed?
  - operations
    - what operations are provided??